AN APPLICATION-LEVEL SOFTWARE WATCHDOG TIMER

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] Not applicable.

STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR DEVELOPMENT

[0002] Not applicable.

BACKGROUND OF THE INVENTION

Field of the Invention

[0003] The present invention generally relates to watchdog timers for personal computer systems. More specifically, the preferred embodiment relates to the use of a software watchdog timer to monitor the uptime of individual applications running on a computer system.

Background of the Invention

[0004] Watchdog circuits are rather common in modern computer systems. A watchdog circuit is one way of creating a stable computing platform. In fact, when one speaks of a stable, robust computer system, the watchdog circuit is indirectly one of the reasons that the system has these attributes. Computer designers rely on the watchdog circuit to reset the system in the unfortunate event something goes wrong. If a computer system hangs or locks up, the watchdog circuit can perform a number of tasks, including logging error information, checking memory, and rebooting the system so the computer will be up and running again in a short amount of time.

[0005] A watchdog circuit typically is a timing circuit that measures a certain system activity or activities. If the system activity does not occur within a prescribed timer period, the watchdog

circuit generates an output signal indicating that the activity has not occurred. In its simplest form, the watchdog timer insures that the system is operational. Modern watchdog circuits are capable of performing a variety of tasks, but the heart of a watchdog timer is essentially just a counter. The timer continually counts up or down using the system clock towards a predetermined value until one of two things happen. First, the counter can be cleared so that the amount of time required to count to the predetermined value is pushed back to the maximum value. For example, if a timer counts from a maximum value of 300 seconds towards a minimum value of zero seconds, then when the timer is cleared, the clock will revert back to the maximum value and continue counting down from 300 seconds. The clear command (sometimes referred to as "hitting the watchdog") is typically issued by the operating system (OS). Programmers will insert commands in the OS code instructing the OS to periodically hit the watchdog. Thus, as long as the OS is operating as intended, the watchdog timer will be cleared periodically and the timer never reaches the predetermined value.

[0006] The second thing that may happen as the watchdog timer is running is that the counter actually does reach the predetermined value. This obviously occurs if the watchdog is never hit and the timer is never cleared. In this case, the watchdog timer will issue a reset command to the system and the computer will reboot. This type of automatic recovery is particularly helpful in unmanned computer systems. Obviously, if a user is working at a computer system and the OS becomes unresponsive, the user can initiate the reset procedure themselves. If, on the other hand, the computer is generally unmanned and working as a server in a computer network, it may not be readily obvious that the computer has ceased normal operations. The first person affected by such a condition will likely be a network user who discovers that they can't access a network database

or perhaps their email. Thus, if a server becomes inoperative, the watchdog timer guarantees that the system will be up and running again in a short amount of time.

[0007] In their present configuration, conventional watchdog timers are certainly useful for their intended purpose. However, there are a number of drawbacks that can be improved upon by a more modern approach. From the perspective of server customers, the health of the OS is not necessarily the most important aspect of a network server. More often than not, a server actually exists to run a specific application and the proper operation of that application is the most important goal for the customer. Thus, if the key application or applications cease operation, but the OS effectively continues, the system will never reset and the customer experiences unwanted downtime.

[0008] Another problem with conventional systems is that the fix for a system lock-up is a full system reset or reboot operation. A more efficient solution to this problem is to first restart the failed application. The time required to end an application process and subsequently restart of that application is much less than the time required to reset the entire system. If the application is successfully restarted, the end result is a decrease in downtime. If, however the OS is unresponsive as well, the conventional watchdog timer will still recover the application by forcing a system reset. In either case, the minimum required downtime is achieved.

[0009] It is desirable therefore, to develop an application-level watchdog timer that is capable of monitoring key applications and reviving those applications in the event the applications become unresponsive. The application-level watchdog timer may work in conjunction with a system level watchdog timer to provide a staggered level of protection that may advantageously improve computer server uptime.

BRIEF SUMMARY OF THE INVENTION

[0010] The problems noted above are solved in large part by a software implementation of an application watchdog comprising a restart service operating in the user mode of a computer operating system and a watchdog driver operating in the kernel mode of the computer operating system. The driver includes a system thread configured to monitor a plurality of user applications that operate in the user mode of the computer operating system. The watchdog driver also provides a first input/output control (IOCTL) signal interface for communicating control signals between the watchdog driver and one of the user applications and a second IOCTL signal interface for communicating control signals between the watchdog driver and the restart service. Lastly, a communication interface is provided for coordinating timer events with the operating system scheduler. Each timer event corresponds to one of the applications and indicates when the application is presumed to be unresponsive.

[0011] If the system thread does not receive a message from an application within an allotted period of time, the timer event alerts the watchdog driver that the allotted time has elapsed and the watchdog driver signals the restart service to restart that application. If the system thread does receive a message from one the applications, the timer event corresponding to that application is updated to reflect the current time plus the allotted period of time. The restart service may also be configured to perform a system reset. Other functions that may be performed by the restart service include: user notification, error logging, and multiple application reset. In addition, the plurality of applications may be prioritized by a computer user to permit varying levels of watchdog protection.

[0012] In the preferred embodiment, the messages from the applications are sent periodically by the applications and directed specifically to the watchdog driver. The messages are preferably sent

to the watchdog driver via a message passing interface between the user mode and kernel mode. The message passing interface is preferably implemented as shared memory queues.

[0013] Initialization of the watchdog driver involves loading the watchdog driver as the operating system loads following a computer system boot. During driver initialization, an initial input/output control (IOCTL) signal interface is loaded and created to establish the message passing interface. A second IOCTL signal interface for communication with the reset service is also created. The restart service is initialized by loading the reset service in the kernel mode of the computer operating system and calling the watchdog driver via the second IOCTL signal interface to verify communication with the watchdog driver.

[0014] The computer application is initialized by linking the application with a dynamic link library and calling the watchdog driver via the dynamic link library to validate the message passing interface. Once this is completed, application information such as the relevant location and process identification is sent to the watchdog driver. The driver sets up timer events for that application and forwards the application information to the reset service. The reset service is then capable of locating and resetting a given application in the event that application becomes unresponsive.

BRIEF DESCRIPTION OF THE DRAWINGS

[0015] For a detailed description of the preferred embodiments of the invention, reference will now be made to the accompanying drawings in which:

[0016] Figure 1 shows a simple computer network comprising a computer system in which the preferred embodiment may be implemented;

[0017] Figure 2 shows a block diagram of a computer system in which the preferred embodiment may be implemented;

[0018] Figure 3 shows a schematic displaying the hardware and software layer architecture of the preferred embodiment; and

[0019] Figure 4 shows a flow chart describing the initialization and operation of the preferred embodiment.

NOTATION AND NOMENCLATURE

[0020] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate, computer companies may refer to a component by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to...". Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0021] Turning now to the figures, Figure 1 shows an example of a simple computer network 10 comprising a plurality of computers. At least one of the computers 20 operates as a central server providing data to the other node computers 100, 120, which are connected to the same network 10. The central server 20 is coupled to the first computer 100 and the second computer 120 by network connections 122. Various other network components such as hubs, switches, modems, and routers may be included in the network 10, but are not shown in Figure 1. It is envisioned server 20 incorporates the preferred embodiment of the invention. Computers 100, 120 may preferably be

"client" computers and may also implement the preferred embodiment. Although a client/server configuration is shown, the computer network may also be an enterprise network, a peer network, a wide area network, a web network or any other suitable network configuration.

[0022] The central server 20 preferably includes at least one input device such as a keyboard 30 and at least one output device such as a monitor 40. Other I/O devices such as a mouse, printer, keyboard, and speakers are certainly permissible and are perhaps desirable peripheral components.

[0023] Users working on computers 100, 120 may remotely access data such as file databases or software applications located on the server 20. Alternatively, software applications may be loaded and run directly on the computers 100, 120, but licenses for the authorized use thereof are located on the central server 20. In either event, if a key application that is needed to provide data from the central server 20 to the network computers 100, 120 becomes unresponsive, that data will become unavailable and users on the network will be inconvenienced.

[0024] It can be appreciated therefore, that the ability to restart a failed application without rebooting the operating system of the server 20 provides certain advantages. The biggest advantage derives from the fact that the total time required to end a process for a key application and restart that application is much less than the time required to reboot the server 20 on which that application is run. The preferred system ensures that the network users are not inconvenienced for an unreasonably lengthy period of time.

[0025] Referring now to Fig. 2, a representative computer system is illustrated. It is noted that many other representative configurations exist and that this embodiment is described for illustrative purposes. For the following discussion, the computer system of Fig. 2 is assumed to represent server computer 20, but one of skill in the art will recognize that the invention may be implemented as part of any computer system. The computer system 20 of Fig. 2 preferably

includes a CPU 202 coupled to a bridge logic device 206 via a CPU bus 203. The bridge logic device 206 is sometimes referred to as a "North bridge" for no other reason than it often is depicted at the upper end of a computer system drawing. The North bridge 206 also couples to a main memory array 204 by a memory bus 205, and may further couple to a graphics controller 208 via an accelerated graphics port (AGP) bus 209. The graphics controller 208 drives the video display 40. The North bridge 206 couples CPU 202, memory 204, and graphics controller 208 to each other and to various peripheral devices in the system through a primary expansion bus (BUS A) such as a PCI bus or an EISA bus. Various components that comply with the bus protocol of BUS A may reside on this bus, such as an audio device 214, a modem 216, and a network interface card (NIC) 217. NIC 217 is coupled to a network 218 for communication with other computers. The above components may be integrated onto the motherboard as presumed by Fig 2, or they may be plugged into expansion slots 210 that are connected to BUS A.

[0026] If other, secondary, expansion buses are provided in the computer system 20, as is typically the case, another bridge logic device 212 is used to couple the primary expansion bus (BUS A) to the secondary expansion bus (BUS B). This bridge logic 212 is sometimes referred to as a "South bridge" reflecting its location vis-à-vis the North bridge 206 in a typical computer system drawing. An example of such bridge logic is described in U.S. Patent No. 5,634,073, assigned to Compaq Computer Corporation. Various components that comply with the bus protocol of BUS B may reside on this bus, such as hard disk controller 222, Flash ROM 224, and I/O Controller 226. Slots 220 may also be provided for plug-in components that comply with the protocol of BUS B.

[0027] The I/O controller 226 typically interfaces to basic input/output devices such as a floppy disk drive 228, a keyboard 30, a mouse 232, a parallel port, a serial port, and, if desired, various

other input switches such as a power switch and a suspend switch (not shown). The I/O controller 226 may incorporate a counter and a Real Time Clock (RTC) to track the activities of certain components such as the hard disk 222 and the primary expansion bus. Alternatively, the clock functions may reside on the Advanced Server Management (ASM) unit 230. The ASM unit 230 includes a system watchdog of the type that is found in many conventional computer systems. An example of such a watchdog is the Automatic Server Recovery (ASR) watchdog found in some Compaq Computer Corporation servers.

[0028] Referring now to Figure 3, a schematic showing the system architecture of the preferred embodiment is shown. The preferred embodiment is described for, but not limited to, a Windows NT environment. The three main levels shown in Figure 3 represent the hardware/software protection layers in a conventional computer system running the Windows NT operating system. The NT environment provides two software protection levels: Ring 0 and Ring 3. Other systems may provide up to 4 or more protection levels. The Ring 0 protection level, sometimes called the kernel mode or supervisor mode, is the most highly protected ring in which an application or service can run. The Ring 3 protection level, sometimes called the application level or user mode, is the least protected ring. Applications running in Ring 3 cannot physically access memory space in the more highly protected Ring 0 layer. Any communication between applications running in Ring 3 and services in Ring 0 must use a message passing service. This design prevents user applications from interfering with the core NT operating system.

[0029] Also shown in Figure 3 is a Hardware layer, which represents the physical computer system hardware such as the CPU, timer devices, and watchdog devices. For the purposes of illustrating the preferred embodiment, Figure 3 shows only the applicable timer device 300, which may be a RTC device or other on-board clock. Also included in Figure 3 is a Hardware

Abstraction Layer (HAL) 310, which is used to prevent hardware dependence and provide an isolation layer between the hardware and software. The HAL operates at the Ring 0 level and translates low-level operating system functions into instructions understandable by the physical system hardware.

[0030] An operating system kernel scheduler 320 is also found in conventional NT architectures. This scheduler dispatches interrupts and performs kernel mode process and thread scheduling. The scheduler 320 uses the OS clock to perform this scheduling. Since the hardware clock is read by the OS at boot time, the OS clock is theoretically identical to the clock from the hardware device 300. Hence, the scheduler 320 operates using the clock signal from the hardware timer 300 by way of the HAL 310.

[0031] Another aspect of Figure 3 that is common to conventional NT system architectures is the location and execution of a user application 330 in the Ring 3 protection layer. As discussed above, the protection levels are set up to ensure a stable operating system environment. In order to provide access to OS functions and data structures, a set of dynamic link libraries (DLL) 340 are used as extensions to the applications. The application 330 and DLL 340 are typically linked at application load time. Furthermore, a message passing interface is used to permit communication between the application 330 in the application layer and kernel mode drivers in the Ring 0 layer. In Figure 3, the shared memory queues 350 perform the message passing function as well as manage any asynchronous inter-layer timing differences.

[0032] The above described architecture will now be supplemented with a description of the unique aspects and advantages of the preferred embodiment. Among the required components is a kernel mode driver 360 with a system thread 370. The system thread 370 processes information and communicates with the shared memory queues 350 situated between the application 330 and

driver 360. A restart service agent 380 is also incorporated to restart desired applications. The restart service may also be configured to restart the entire system much like the hardware watchdog timer, but for the preferred embodiment, the restart service 380 maintains reset control over applications only.

[0033] The application watchdog driver 360 also uses I/O control calls (IOCTL), which are user-defined requests and instructions passed to and from kernel mode drivers. In accordance with the preferred embodiment, the watchdog driver 360 establishes an initial IOCTL interface 390 that establishes the appropriate message passing interface 350 and a run-time IOCTL signal interface 395 for communication with the application restart service. The initialization and use of the IOCTL interfaces 390, 395 are described below.

[0034] Referring now to Figure 4, a simplified flow chart describing the initialization and operation of the preferred embodiment is shown. The following description includes references to the watchdog system architecture as shown in Figure 3. The START procedure 400 begins during a computer system reset. This reset may be a cold boot, warm boot, or perhaps even a system reset initiated by the system level watchdog timer. After the computer completes the boot operation and executes the POST operation, the operating system will load and initialize 410. During OS initialization 410, the kernel mode watchdog driver 360 will load and create an initial IOCTL 390 interface with commands for establishing the message passing interface. The watchdog driver 360 will also establish an IOCTL signal interface 395 for communication with the restart service 380. As part of the restart service initialization 420, the watchdog restart service 380 will call the watchdog driver 360 to verify existence and operation of the IOCTL signal interface 395. Once the restart service 380 is established, the key user application 330 is started and initialized 430. Once the application is linked to an appropriate DLL 340, the application will call into the DLL

340, which in turn, will make the initialization IOCTL calls 390 into the watchdog driver 360 to establish a connection through the message passing interface or shared memory queues 350. Once this interface is established, no further IOCTL calls will be required. The initialization IOCTL calls 390 will likely have pointers and callback addresses associated with the user application. In response to the initialization IOCTL calls 390, the watchdog driver 360 will, in turn, use the restart service IOCTL interface 395 to update the restart service 380 with application information such as process id's and address information. Thus, the restart service will have information needed to end a particular process and restart the application should the need arise.

[0035] During runtime operation the user application sends messages periodically through the interface 350. The watchdog driver system thread 370 will asynchronously monitor the interface 350 for periodic messages from the application 440. If the watchdog driver 360 does not detect a message from the application 330 for a predetermined period of time, the driver 360 will signal the restart service 380 to terminate and restart the application 450. The terminate procedure may consist of killing the appropriate process or ending a task as provided for by the operating system. Otherwise, the application watchdog system simply continues monitoring the shared memory queues 350 for the periodic messages until the application 330 is manually closed down or the computer system is shut down 460.

[0036] As discussed above, the restart service 380 may also be configured to provide system restarts if a system administrator feels that is the appropriate reaction. It is envisioned that an alternative embodiment may provide a user interface that allows the user to choose the level of reset capability. For example, the user may be able to choose between application, operating system, or full system reset. Also, the restart service may also perform other functions such as error logging or local/remote user notification.

[0037] The watchdog driver 360 preferably includes variables for the various applications to be monitored as well as how frequently messages from the applications should be expected. The driver 360 preferably uses this information and works in conjunction with the OS scheduler 320 to set up alarms or events that will notify the application watchdog driver 360 if a particular application has stopped responding. The timer events represent the end of some allotted period of time during which the watchdog driver 360 should expect a message from the application 330 under normal operating conditions. These timer events are reset or postponed as new messages are detected from the application 330. Thus, detecting a message serves the same function as hitting the watchdog in a conventional watchdog timer. If the timer event actually occurs, this means the application 330 has not delivered a message to the shared memory queues 350 for some time and the application 330 is presumed dead or unresponsive. The application watchdog driver 360 then responds to the timer event by directing the restart service 380 to initiate the appropriate reset procedure.

[0038] It is envisioned that the periodic signals sent by the application will be initiated by commands embedded in the computer application software. These commands will be directed at the shared memory queues 350 for the purpose of resetting the application watchdog timer events. It is feasible however, that the commands be sent as part of normal communication with other parts of the computer including the CPU, system memory, or the OS. In this case, the watchdog driver system thread 370 acts as a passive observer checking for activity from the application 330. Other embodiments in accordance with the above teachings are certainly feasible.

[0039] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. For example,

since the watchdog driver 360 is capable of monitoring several applications, the watchdog system may be configured to provide a user interface to establish priority among the applications. For instance, some sort of policy control may be added that allows the alarm timer events to be delayed more for one application compared to others. This will provide some measure of certainty to ensure that an application has hung before it is restarted. It is intended that the following claims be interpreted to embrace all such variations and modifications.